Data och Informationsteknik / Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Erland Holmström Göteborg 9 Mar 2017

Examination i

Datastrukturer och Algoritmer IT, TDA416

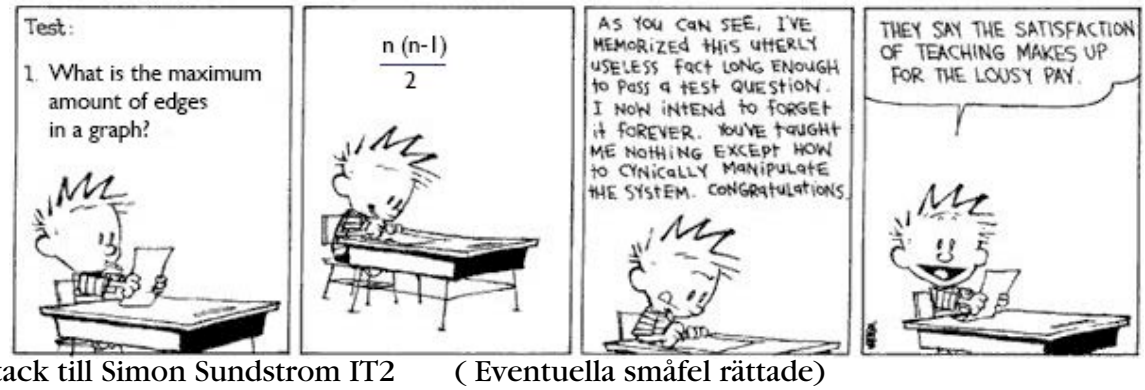
Dag: Lördag Datum: 2017-03-11 Tid: 8.30-13.30 (OBS 5 tim) Rum: M

Ansvarig lärare:	Erland Holmström tel. 1007, mobil 0708-710 600
Besökande lärare:	Erland Holmström tel. 1007,
Resultat:	Skickas med mail från Ladok.
Lösningar:	Läggs eventuellt på hemsidan.
Tentagranskning:	Tentan finns efter att resultatet publicerats på vår expedition. Tid för klagomål på rättningen annonseras på hemsidan efter att resultatet är publicerat (eller maila mig så försöker vi hitta en tid).
Betygsgränser:	CTH: 3=28p, 4=38p, 5= 48p, max 60p För betyg 3 skall man ha 10 poäng på del A och10p på del B. Dock kan gränsen variera något beroende på tentans upplägg.
Hjälpmedel:	Bara en sammanfattning av delar av Javas API som kan lånas på tentan eller skrivas ut i förväg, se hemsidan.

Observera:

- Börja med att läsa igenom alla problem så du kan ställa frågor när jag kommer. Jag brukar komma efter ca 1-2 timmar (men det är många salar så det kan bli senare).
- ***Alla svar måste motiveras*** där så är möjligt/lämpligt.
- Skriv läsligt! Rita figurer. Lösningar som är svårlästa bedöms inte.
- Skriv kortfattat och precist.
- Råd och anvisningar som getts under kursen skall följas.
- Program skall skivas i Java, skall vara indenterade och lämpligt kommenterade.
- Börja nya problem på ny sida, dock ej korta delproblem.
- *Lämna inte in tesen med tentan utan behåll den. Lämna inte heller in kladdpapper/skisser.*

Lycka till!



Tipstack till Simon Sundstrom IT2 (Eventuella småfel rättade)

Del A Teori *Du måste motivera även om det inte står explicit.*

Problem 1. Uppvärmning:

Vilken eller vilka av följande påstående är sann(a) och vilken eller vilka är falsk(a) eller svara på frågan. Alla svar skall motiveras.

a) Västafallskomplexiteten för find i ett splay träd är O(n).

b) I en hashtabell tar det alltid konstant tid att slå upp värdet till en nyckel.

c) Antag att `o1.hashCode()` returnerar samma värde som `o2.hashCode()`. Kommer då `o1.equals(o2)` alltid att returnera true?

3p

Problem 2. *Testar: Träd*

Beskriv kortfattat egenskaperna hos följande träd, ange speciellt om dom är ordnade och / eller balanserade eller inte.

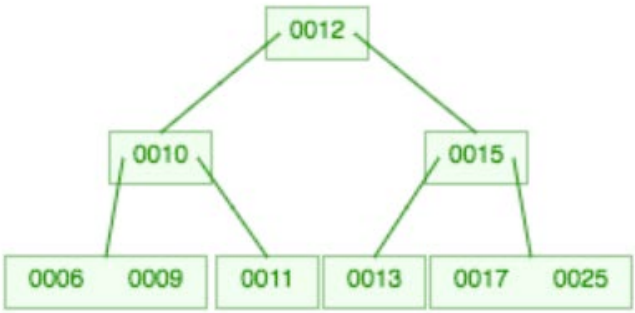
a) binärt sökträd, AVL-träd, splay-träd, B-träd,
samt partiellt ordnat+vänstebalanserat träd (dvs en heap).

b) Sätt in talen $4, 9, 2, 8, 5, 6$ i given ordning i ett AVL-träd.. Rita trädet efter varje operation*, även stegen i varje operation vid balanseringen så man kan följa vad du gör. Du bör också motivera varför du gör just de balanseringar du gör.
* Du kan sätta in de 4 första talen innan du ritar trädet.

5+6 (11p)

Problem 3. *Testar: 2-3 träd*

Låt T vara (2, 3) trädet nedan i vilket vi har heltalsnycklar.



Rita och motivera alla steg för att

a) Sätta in elementet 8 i trädet T.

b) Ta bort elementet 13 ur trädet T.

(6p)

Problem 4. *Testar: sortering*

Varför är det viktigt hur man väljer pivotelement i quicksortalgoritmen?
Ge belysande exempel på hur indata, pivotelement-val och komplexitet ser ut när quicksortalgoritmen beter sig som ”sämst” och när den beter sig som ”bäst”.

(För ”bäst” är det lämpligt att beskriva både teoretisk bästa val och praktiskt bra val.)

(4p)

Problem 5. *Testar: Grafer*

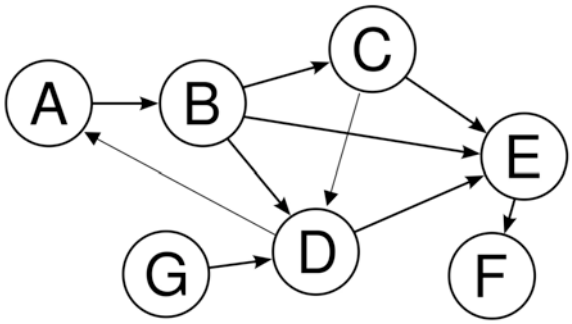
a) Beskriv (dvs rita) hur grafen nedan representeras som en grannlista.

b) Som en grannmatris.

c) Ge en kort (steg 1) beskrivning av djupet först i grafer samt beskriv i vilken ordning noderna i grafen besöks om vi gör en djupet-först genomgång av grafen vid start i nod A. Välj i bokstavsordning om ordningen inte ges av algoritmen tex vid val av en av två möjliga grannar.

d) Ge en kort beskrivning av Dijkstras algoritm. (kort = steg 1; ide och steg 2; pseoudokod). Applicera sedan Dijkstras algoritm på grafen nedan med start i nod B. Gör en tabell över algoritmens fortskridande och skriv ner vad som händer. Du kan använda tabellen sist i tentan om du vill.

Avstånd: (A,B,1), (B,C,3), (B,D,6), (B,E,9), (C,D,2), (C,E,5), (D,A,1), (D,E,2), (E,F,1), (G,D,1).

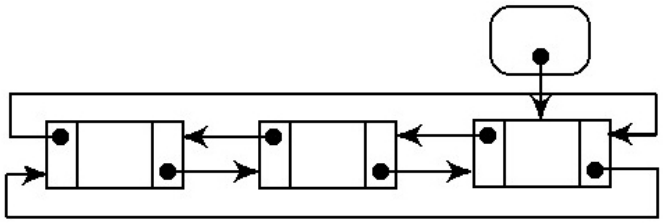


(12p)

Del B Implementeringar och algoritmer

Problem 6. Testar: länkade listor, pekarhantering, iteratorer

Antag att vi vill implementera en samling som en dubbellänkad cirkulär struktur.



För att hålla reda på början/slutet så har vi en variabel ”last” som pekar på det sista elementet i samlingen (som då är ett element före det första eftersom den är cirkulär). Om vi tänker oss att listan i figuren har sin riktning åt höger så är det vänstra elementet det första och det högra är det sista elementet.
Vi startar med deklarationerna som du kan se på sista sidan för en **D**ouble **L**inked **C**ollection. Där ser du signaturer för alla metoder som skall skrivas. När du skriver en metod kan du anta att dom andra metoderna i klassen finns. Du får inte ändra i specifikationen. Pekarna åt vänster kallas ”prev” och de åt höger kallas next i koden i DLC.

a) Antag att det inte finns någon variabel som håller reda på antalet element i samlingen. Skriv metoden `int size()` som beräknar detta.

b) Skriv metoden `boolean add(E elem)` som lägger till ett element sist i samlingen.

c) Skriv metoden `boolean remove(Object o)` som tar bort den första instansen av `o` i samlingen. Det är lämpligt att dela upp arbetet så att `remove` hittar noden som skall tas bort och anropar metoden `removeThisNode` som tar bort den.

d) Vad är komplexiteten för metoderna ovan. Motivera.

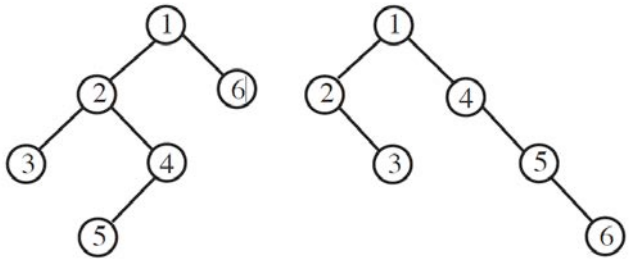
(14p)

vänd

Problem 7. *Testar: rekursion över träd.*

När är träd lika egentligen?

Träden nedan betraktas normalt inte som lika eftersom man brukar kräva att både strukturen hos träden och innehållet i noderna är lika. Men om vi tittar på preorder-genomlöpningen av träden så är den lika nämligen 1, 2, 3, 4, 5, 6.



Antag att vi har deklarationerna nedan för ett binärt träd.

a) Utöka klassen BST nedan med en *effektiv* publik metod `String toString()` som returnerar nodernas innehåll i preorder i form av en sträng formaterad enligt nedan. Nodernas innehåll är heltal. Komplexitet?

   [1 2 3 4 5 6]

b) Utöka också med en metod `boolean equals(Object o)` som avgör om det implicita trädet (this) är lika med o avseende preordergenomlöpningen. (Metoden skall fungera även om vi i framtiden ärver den här klassen.)

(10p)

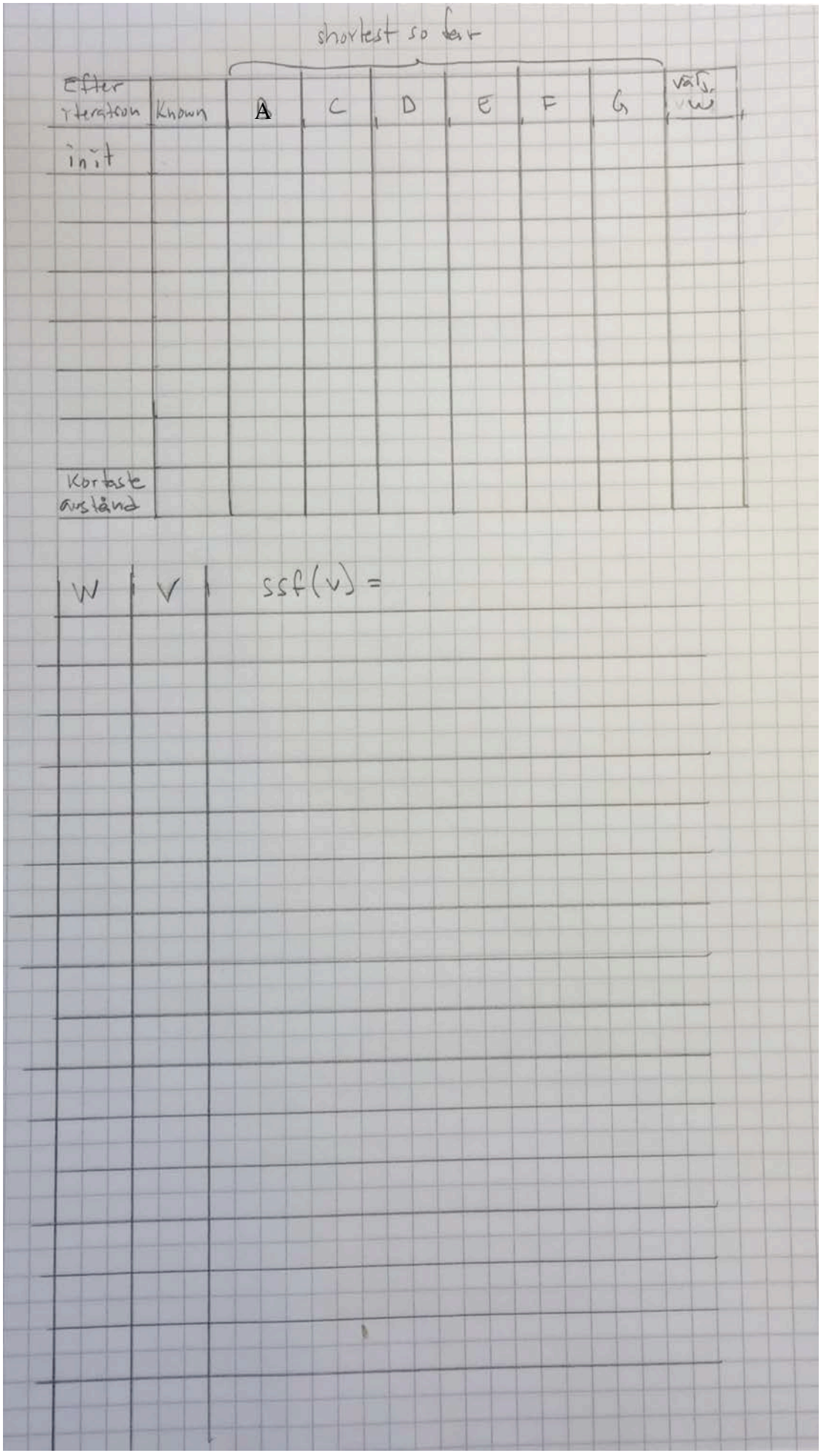
```
public class BST<E extends Comparable<? super E>> {
  protected Entry root;
  // ========== ========== ========== ==========
  protected class Entry {
    public E      element;
    public Entry  left, right;
  ... konstruktorer för Entry klassen utelämnade
  }
  // ========== ========== ========== ==========
  // the obvious
  public int size() {...}
  /**
  * Create an iterator for elements in the tree in preorder.
  * @return the created iterator.
  */
  public Iterator<E> iterator() {
    return new BSTPre_Iterator();
  }  // end iterator
... resten av klassen BST utelämnad
}
```

```
public class DLC<E> extends AbstractCollection<E> {
  protected Node last; // pointer to the end of the list

  protected class Node {
    Node next, prev;
    E elem;
    public Node(Node next, Node prev, E elem) {
      this.next = next;
      this.prev = prev;
      this.elem = elem;
    }
    public Node(E elem) {
      this.next = this; // pekar på sig själv
      this.prev = this;
      this.elem = elem;
    }
  } // end Node

  public DLC() {...}
  public int size() { ... }
  public boolean add(E elem) { ... }
  public boolean remove(Object o) { ... }
  private void removeThisNode(Node p) { ... }

  public String toString() {
    StringBuilder str = new StringBuilder();
    if (last==null) {
      return "<>";
    } else {
      Node p = last.next;
      str.append("<" + p.elem + ">");
      p = p.next;
      while (p!=last.next) {
        str.append(",<" + p.elem + ">");
        p = p.next;
      }
    }
    return str.toString();
  }
  // ========= ========= ========= =========
  public Iterator<E> iterator() {
    return new DLCIterator();
  }
  protected class DLCIterator implements Iterator<E> {
    public DLCIterator(){ ... }
    public boolean hasNext() { ... }
    public E next( ) { ... }
    public void remove() { ... }
    final void checkForComodification() { ... }
} // end DLC
```


shortest so far
Efter iteration
Known
A
C
D
E
F
G
väg. w
init
Kortaste avstånd
w
v
ssf(v) =